(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 136 912 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.09.2001 Bulletin 2001/39

(51) Int Cl.7: G06F 11/25

(21) Application number: 01101044.4

(22) Date of filing: 18.01.2001

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 22.03.2000 US 533115

(71) Applicant: Agilent Technologies Inc.
a Delaware Corporation
Palo Alto, CA 94303 (US)

(72) Inventors:
- Barford, Lee A.
San Jose, CA (US)
- Smith, David R.
San Jose, CA (US)

(74) Representative: Barth, Daniel
c/o Agilent Technologies Deutschland GmbH,
Herrenbergerstrasse 110
71034 Böblingen (DE)

(54) Probalistic diagnosis, in particular for embedded and remote applications

(57) Disclosed is a diagnosis engine for diagnosing a device with a plurality of components. States of the components are assumed to be probabilistically independent, for computing the probability of any particular set of components being bad and all others being good. States of shared functions, applicable for testing the functionality of some components in the same way, are assumed to be probabilistically independent given component states, for computing the probability of any particular set of shared functions being failed and another particular set of shared functions being passed given that a particular set of components are bad. States of tests applicable on the device are assumed to be probabilistically independent given component and shared function states, for computing the probability of any particular set of tests being failed and another particular set of shared functions being passed given that a particular set of components are bad, and the rest are good, and a particular set of shared functions are failed, and the rest are passed. The diagnosis engine receives test results of a set of tests on the device where at least one test has failed, and a model giving the coverage of the tests on the components of the device and information describing probabilistic dependencies between the tests. The diagnosis engine comprises means for setting or specifying a number N of components which may be simultaneously bad, and computing means for computing the likelihood that each of subsets of the components with size less than or equal to N are the bad components, whereby the computation is substantially exact within floating point computation errors.

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND OF THE INVENTION

[0001] The present invention relates to monitoring, detecting, and isolating failures in a system, and in particular to tools applied for analyzing the system.

[0002] "To diagnose" means to determine why a malfunctioning device is behaving incorrectly. More formally, to diagnose is to select a subset of a predetermined set of causes responsible for the incorrect behavior. A diagnosis must both explain the incorrect behavior and optimize some objective function, such as probability of correctness or cost of incorrect diagnosis. The need to diagnose is a common reason to measure or to test.

[0003] The diagnosis of an engineered device for the purpose of repair or process improvement shall now be regarded. This is in contrast to, say, a distributed computer system containing software objects that may be created or destroyed at any time. It is assumed that the device consists of a finite number of replaceable components. Failures of the device are caused only by having one or more bad components. What shall be called herein "diagnosis" is often called "fault identification". When presented with a failed device, a technician or a computer program (sometimes called a "test executive") will run one or more tests. A technician familiar with the internal workings of a failing device must interpret the test results to identify the bad components.

[0004] Expert systems have been used for diagnosing computer failures, as described e.g. by J.A. Kavicky and G. D. Kraft in "An expert system for diagnosing and maintaining the AT&T 3B4000 computer: an architectural description", ACM, 1989. Analysis of data from on-bus diagnosis hardware is described in Fitzgerald, G.L., "Enhance computer fault isolation with a history memory," IEEE, 1980. Fault-tolerant computers have for many years been built with redundant processing and memory elements, data pathways, and built-in monitoring capabilities for determining when to switch off a failing unit and switch to a good, redundant unit (cf. e.g. US-A-5,099,485).

[0005] Prior diagnostic systems for determining likely failed components in a system under test (SUT) include model-based diagnostic systems. A **model-based diagnostic system** may be defined as a *diagnostic system that renders conclusions about the state of the SUT using actual SUT responses from applied tests and an appropriate model of correct or incorrect SUT behavior as inputs to the diagnostic system.* Such a diagnostic system is usually based upon computer-generated models of the SUT and its components and the diagnostic process.

[0006] Model-based diagnostic systems are known e.g. from W. Hamscher, L. Console, J. de Kleer, in *'Readings in system model-based diagnosis'*, Morgan Kauffman, 1992. A test-based system model is used by the Hewlett-Packard HP Fault Detective (HPFD) and described in *HP Fault Detective User's Guide,* Hewlett-Packard Co., 1996.

[0007] US-A-5,808,919 (Preist et al.) discloses a model-based diagnostic system, based on functional tests, in which the modeling burden is greatly reduced. The model disclosed in Preist et al. employs a list of functional tests, a list of components exercised by each functional test along with the degree to which each component is exercised by each functional test, and the historical or estimated a priori failure rate for individual components.

[0008] US-A-5,922,079 (Booth et al.) discloses an automated analysis and troubleshooting system that identifies potential problems with the test suite (ability of the model to detect and discriminate among potential faults), and also identifies probable modeling errors based on incorrect diagnoses.

[0009] EP-A-887733 (Kanevsky et al.) discloses a model-based diagnostic system that provides automated tools that enable a selection of one or more next tests to apply to a device under test from among the tests not yet applied based upon a manageable model of the device under test.

[0010] In the above three model-based diagnostic systems, a diagnostic engine combines the system-model-based and probabilistic approaches to diagnostics. It takes the results of a suite of tests and computes - based on the system model of the SUT - the most likely to be failed components.

[0011] The diagnostic engine can be used with applications where a failing device is to be debugged using a predetermined set of test and measurement equipment to perform tests from a pre-designed set of tests. Using test results received from actual tests executed on the SUT and the system model determined for the SUT, the diagnostic engine computes a list of fault candidates for the components of the SUT. Starting, e.g., from a priori failure probabilities of the components, these probabilities may then be weighted with the model information accordingly if a test passes or fails. At least one test has to fail, otherwise the SUT is assumed to be good.

[0012] An embedded processor is a microprocessor or other digital computing circuit which is severely limited in computing power and/or memory size because it is embedded (i.e., built in to) another product. Examples of products typically containing embedded processors include automobiles, trucks, major home appliances, and server class computers (which often contain an embedded maintenance processor in addition to the Central Processing Unit(s). Embedded processors typically have available several orders of magnitude less memory and an order of magnitude or two less computing power than a desktop personal computer. For example, a megabyte of memory would be a large amount for a home appliance. It is desirable to enable such an embedded processor in such a product to diagnose failures of the product. A diagnosis engine providing such a capability shall be called an **embedded diagnosis engine.**

[0013] It is possible to perform probabilistic diagnosis by various **heuristic methods**, as applied by the aforementioned HP Fault Detective product or US-A-5,808,919 (Preist et al.). Heuristics by nature trade off some accuracy for reduced computation time. However, the HP Fault Detective typically requires 4 to 8 megabytes of memory. This is can be a prohibitive amount for an embedded diagnosis engine.

[0014] Another method for solving the problem is **Monte Carlo simulation.** Although the Monte Carlo simulation method can be made arbitrarily accurate (by increasing the number of simulations), the simulation results must be stored in a database that the diagnosis engine later reads. It has been shown that, even when stored in a space-efficient binary format, this database requires 2-6 megabytes for typical applications. This is too much for embedded application and would be a burden on distributed application where the database might have to be uploaded on a computer network for each diagnosis.

[0015] A common way of building a probabilistic diagnostic system is to use a **Bayesian network** (cf. Finn V. Jensen: "Bayesian Networks", Springer Verlag, 1997). A Bayesian network is a directed acyclic graph. Each node in the graph represents a random variable. An edge in the graph represents a probabilistic dependence between two random variables. A source (a node with no in-edges) is independent of all the other random variables and is tagged with its a priori probability. A non-source node is tagged with tables that give probabilities for the value of the node's random variable conditioned on all of the random variables upon which it is dependent.

[0016] The computation on Bayesian networks of most use in diagnosis is called belief revision. Suppose values of some of the random variables (in the context of herein, the results of some tests) are observed. A belief revision algorithm computes the most likely probabilities for all the unobserved random variables given the observed ones. Belief revision is NP-hard (cf. M. R. Garey and D. S. Johnson: "Computers and Intractability: A guide to the theory of NP-completeness", W. H. Freeman and Co., 1979), and so all known algorithms have a worst-case computation time exponential in the number of random variables in the graph.

[0017] Bayesian networks used for diagnosis are constructed with random variables and their dependencies representing arbitrary cause-and-effect relationships among observables such as test results, unobservable state of the device under diagnosis and its components, and failure hypotheses. The graph can grow very large and have arbitrary topology. For example, an experimental Bayesian network used by Hewlett-Packard for printer diagnosis has over 2,000 nodes. The complexity of such networks creates two difficulties:

- all of the conditional probabilities for non-source nodes must be obtained or estimated, and
- local changes to topology or conditional probabilities may have difficult-to-understand global effects on diagnostic accuracy.

[0018] In other words, the use of a large Bayesian net of arbitrary topology for diagnosis has somewhat the same potential for supportability problems as do rule-based diagnostic systems.

SUMMARY OF THE INVENTION

[0019] It is an object of the invention to provide an improved probabilistic diagnosis, which may also be applicable for embedded and/or remote applications. The object is solved by the independent claims. Preferred embodiments are shown by the dependent claims.

[0020] The present invention provides a diagnosis engine: a tool that provides automatic assistance e.g. to a technician at each stage of a debugging process by identifying components which are most likely to have failed.

[0021] The major advantage of the invention over other diagnosis engines is that it can be provided with a small memory footprint: both code and runtime memory requirements are small, growing only linearly with the model size.

[0022] The invention is preferably written entirely in Java (cf. e.g. James Gosling, Bill Joy, and Guy Steel: The Java Language Specification, Addison Wesley, 1996) and preferably uses only a few classes from the Java standard language library packages. These features make the invention in particular well suited to embedded and distributed applications.

[0023] The invention is meant to be used on applications where a failing device is to be debugged using a predetermined set of test and measurement equipment to perform tests from a pre-designed set of tests. For the purposes of herein, a test is a procedure performed on a device. A test has a finite number of possible outcomes. Many tests have two outcomes: pass and fail. For example, a test for repairing a computer may involve checking to see if a power supply voltage is between 4.9 and 5.1 volts. If it is, the test passes. If it isn't, the test fails. Tests may have additional outcomes, called failure modes. For example, a test may involve trying to start an automobile. If it starts, the test passes. Failure modes might include:

- the lights go dim when the key is turned, and there is no noise from under the hood,

- the lights stay bright when the key is turned, and there is the noise of a single click,
- the lights stay bright when the key is turned, there is a click, and the starter motor turns, but the engine doesn't turn over, and so forth.

[0024] The set of all tests available for debugging a particular device is called that device's test suite. Many applications fit these definitions of debugging and of tests. Examples are:

- computer and electronics service and manufacturing rework,
- servicing products such as automobiles and home appliances, and
- telephone support fits the model, if we broaden the idea of "test" to include obtaining answers to verbal questions.

[0025] Given:

- a set of tests on a physical object (e.g., Test 1 = pass, Test 2 = fail, Test 3 = pass, etc.) where at least one test has failed, and
- a model giving the coverage of the tests on the components (e.g., field replaceable units) of the object and information describing probabilistic dependencies between tests,

the invention outputs a probabilistic diagnosis of the object, that is, a list, each element of which contains:

- a list of one or more components, and
- the likelihood or probability that those components are the bad components. (Likelihood is un-normalized probability. That is, probabilities must sum to one but likelihoods need not.)

[0026] Most automated diagnosis systems provide simply a list of possible diagnoses without weighting by probability. Having probabilities is particularly desirable in applications where the number of field replaceable units (FRU) is small, because then most FRUs will be found to be possible diagnoses much of the time. The probabilities also give technicians an opportunity to apply their own expertise.

[0027] The invention allows handling multiple component failures. No distinction is made between single and multiple faults.

[0028] The invention combines the model-based (cf. W. Hamscher, L. Console, and J. de Kleer: Readings in model-based diagnosis, Morgan Kauffman, 1992) and probabilistic approaches to diagnostics. The invention uses the same test-based model as by the aforementioned HP Fault Detective or in US-A-5,808,919 (Preist et al.). This model describes probabilistic relationships between tests and the components that they test in a manner intended to be accessible to engineers who write tests. Features of this model can be preferably:

- a two-level part-whole hierarchy: names of components (field-replaceable units) and of their sub-components,
- estimates of a priori failure probabilities of the components,
- the names of the tests in the test suite,
- an estimate of the coverage that each test has on each component, i.e., the proportion of the functionality of the component that is exercised by the test, or more formally, the conditional probability that the test will fail given that the component is bad,
- shared coverages of tests, which are a way of modeling tests that are dependent because they test the functionality of some components in exactly the same way (for example, two tests that access a certain component through a common cable have shared coverage on the cable), and
- a way of specifying failure modes for tests in addition to pass and fail. Failure modes have a name, and two lists of components or sub-components. The first list, called the acquit list, names the components or sub-components that must have some operable functionality in order for the failure mode to occur. The second list, called the indict

list, names the components or sub-components that may be bad if the failure mode occurs. Each entry in the acquit and indict lists also contains an estimate of the amount of functionality of the component that the failure mode exercises.

[0029] Models can be created e.g.:

- Using a model-building graphical user interface (GUI) that comes e.g. with the aforementioned HP Fault Detective. The HP Fault Detective model is read by a program that translates it into a simpler form used internally by the invention, which can be saved as an ASCII file. The invention can load such a file from a file system, from a URL, or from local memory.
- By writing ASCII test Fault Detective Model (.fdm) files, or
- Through a model creation application programming interface (API) in Java.

[0030] The model, together with the rules of mathematical logic, enables to compute the probability that a test will fail if a particular component is known to be bad. More details about these models and the model-building process are disclosed in the co-pending US patent application (Applicant's internal reference number: US 20-99-0042) by the same applicant and in US-A-5,922,079 (Booth et al.). The teaching of the former document with respect to the description of the model and the model-building process are incorporated herein by reference.

[0031] The invention allows computing the probability of a test's failure when given any pattern of components known to be good or bad. The logic formula known as Bayes' Theorem allows running this computation in reverse: given a particular test result, the invention can calculate the probability of occurrence of some particular pattern of component faults and non-faults. The invention, then, enumerates all the possible patterns of component faults/non-faults, evaluating the probability of each pattern given the test result. The pattern with highest probability is selected as the diagnosis.

[0032] Of course, one test is seldom sufficient to make an unambiguous diagnosis. If the test succeeds, it may clear some components, but not indicate the culprit. If it fails, it may indict several components, and other tests are required to clear some or focus suspicion on others. (Here, "clearing" means to knock the computed fault probability way down, and "focusing suspicion" means to raise the probability to the top or near the top.) Handling multiple test results is easy and quick if the tests are independent of each other. But if the tests are not independent, the problem is much more complex. The dependence is modeled by the shared functions. A case-by-case breakdown must be made of all the ways the shared functions might pass or fail and how they affect the joint probabilities of the test results. Then all these influences must be summed, as sketched e.g. in the outline of a diagnosis algorithm (in pseudo-code) as shown below:

```
1. For each possible combination of bad components:
    (a) Set sum to 0.
    (b) For each possible pass/fail combination of shared functions:
        i. Compute the probability of the observed test results.
        ii. Add the probability to sum.
    (c) Calculate likelihood of the combination of bad components given sum (using Bayes' Theorem).
2. Sort the fault likelihoods in descending order.
```

[0033] The algorithm iterates over combinations of failed components and computes the conditional likelihood of each combination given passed and failed tests.

[0034] Clearly, this method can require enormous amounts of computation as it explores all combinations of shared function outcomes for all combinations of faults.

[0035] The mathematical detail how all this is to be accomplished and also how the computational burden is reduced to allow the method to be practical will be shown in great detail in the section 'Detailed Description of the Invention'.

[0036] Any model as used by the invention can be represented by a Bayesian network. The resulting graph is tripartite, consisting solely of sources, sinks, and one level of internal nodes (as shown later). There is one source for each component. There is one sink for each test. Each shared function is represented by one internal node. However, in order to represent test coverage information, the so-called "Noisy-or" (defined and described in detail in chapter 3 of

Finn V. Jensen: "Bayesian Networks", Springer Verlag, 1997) construction must be used. The form of test coverage information is such that the memory-saving technique of divorcing (again, see Chapter 3 of Jensen) cannot be used. This means that the Bayesian network will require an amount of memory exponential in the number of components covered by any test. Even small models exhaust the memory of desktop PC workstations. Clearly this approach is not well suited to embedded or distributed application.

[0037] The class of models as applied by the invention, then, can be viewed as a subclass of the Bayesian networks. The invention diagnosis algorithm can be considered an efficient algorithm for belief revision over this subclass. The high accuracy rate of successful diagnosis with the invention (as shown later) suggests that this subclass is sufficiently powerful to represent practical diagnosis problems. Furthermore, the relatively structured nature of a model in accordance with the present invention may be an advantage when building and supporting a model when compared with free-form construction of a Bayesian network.

[0038] Like Bayesian Networks, the invention computes the component failure likelihoods exactly. Heuristic methods and Monte Carlo simulation compute approximate likelihoods. The runtime performance of the invention is good in practice. It runs about as fast as the aforementioned HP Fault Detective on the same diagnosis problems.

[0039] In a nutshell, the invention is based on the assumptions that:

1. Component states (that is, whether each component is good or bad) are probabilistically independent;

2. Shared function states (that is, whether each shared function is passed or failed) are probabilistically independent given component states; and

3. Test states (that is, whether each test is passed or failed) are probabilistically independent given component and shared function states.

[0040] Assumption 1 is used to compute the probability of any particular set of components being bad and all others being good.

[0041] Assumption 2 is used to compute the probability of any particular set of shared functions being failed and another particular set of shared functions being passed given that a particular set of components are bad.

[0042] Assumption 3 is used to compute the probability of any particular set of tests being failed and another particular set of shared functions being passed given that a particular set of components are bad (and the rest are good) and a particular set of shared functions are failed (and the rest are passed).

[0043] Thus, the invention provides:

1. Means of specifying the component prior probabilities, coverages, and shared function coverages.

2. Means of specifying which tests have passed and which tests have failed. (Some tests may be neither passed nor failed because they were never performed.)

3. Means of specifying how many components may be simultaneously bad. Call this number N.

4. Means of computing the likelihood that each of the subsets of the components with size less than or equal to N are the bad components, whereby

- the computation is exact (to within small floating point computation error); and
- the amount of memory required to perform the computation is preferably only a constant amount larger than the memory required to store the inputs and the outputs. ("A constant amount larger" shall mean an increased amount that is the same independent of the model size and N.)

5. Means of outputting the likelihoods, either

- in human-readable form, or
- as computer data available for further automatic processing.

[0044] Instead of feature #3, a default value (probably 1 or 2) could be built in. This would reduce the flexibility in using the invention without impairing its usefulness much.

[0045] The invention thus allows construction of a diagnosis engine which will require an amount of memory less than the amount of memory required to store the model and the amount of memory required to store the output multiplied by a small factor which is a constant independent of the model and output sizes. This makes such a diagnosis engine well suited to use as an embedded diagnosis engine.

[0046] It is clear that the invention can be partly or entirely embodied by one or more suitable software programs, which can be stored on or otherwise provided by any kind of data carrier, and which might be executed in or by any suitable data processing unit.

DETAILED DESCRIPTION OF THE INVENTION

[0047] **Table 1** shows the notation for items from the coverage-based model, such as components, tests, and shared functions and gives formal definitions for coverage and shared function variability.

[0048] For the sake of simplicity, the term "Equation" shall be used in the following not only for pure mathematical equations but also for mathematical terms to which it is referenced in this description.

[0049] The **components** Φ are random variables that can have the states {good, bad}. The prior probabilities P (c bad) for $c \in \Phi$ are given in the model. Component failures are assumed to be independent as defined in Equation (0.1).

[0050] **Shared functions** are used to express probabilistic dependencies between tests. Shared functions may be thought of as expressing the fact that some functionality is checked by different tests in exactly the same way. The shared functions Ω are random variables with the states {pass, fail}. A shared function s is dependent on the states of the components Φ as shown in Equation (0.2), where the shared function coverages sfcov(s, c) is given in the model. The shared functions are conditionally independent of one another as given in Equation (1).

[0051] Intuitively, a shared function fails if it has coverage on a bad spot in any component on which the shared function has coverage. Formally, the probability of a shared function s failing dependent on the states of all components is defined by Equations (2) and (3).

[0052] Equation (3) means that the computation can be performed while iterating through a sparse representation of sfcov. Each shared function s has a variability, sfvar(s) between 0 and 1. Intuitively, variability of a shared function says how correlated are the failures of the tests that use the shared function when the shared function is failed. A variability of 0 means that all tests that use the shared function will fail if the shared function fails. A variability of 1 means that tests that use the shared function may fail independently of each other if the shared function fails. In this case, the shared function is being used as a modeling convenience. The notion of shared function variability will be formalized below.

[0053] The **tests** ψ are random variables with the states {pass, fail}. Generally, only some of the tests are performed. Let Π be the passed tests. Let φ be the failed tests. A test is dependent on both components and shared functions. The coverages P are defined by Equation (3.1) and given in the model. The shared functions used by a test sfused(t) $\subseteq \Omega$ is also given in the model. Tests are conditionally independent of one another given the states of all components and shared functions as shown in Equation (4).

[0054] If a test uses no shared functions, its probability of failure depends on the component states. Intuitively, a test fails if it has coverage on a bad spot. Formally, the probability of a test t failing, when t uses no shared functions, dependent on the states of all components is defined by Equations (5) and (6). Equation 6 means that the computation can be performed by iterating through a sparse representation of cov.

[0055] When a test uses shared functions, it can also fail if any of those shared functions fail. Let's assume Equation (6.1). The conditional probability of test success is then given in Equation (7) and the conditional probability of test failure is its complement as shown by Equation (8).

[0056] All probabilistic dependencies between the three sets of random variables Φ, Ω, and ψ are given in the aforementioned Equations. Otherwise the random variables are independent. Thus, the dependencies among the random variables could be represented by a **Bayesian Network** where the sources of the directed acyclic graph (DAG) are the components Φ and the sinks are the tests ψ. Each nonzero entry in cov, say cov(t, c), results in an edge from component node c to test node t. Each nonzero entry in sfcov, say sfcov(s, c) results in an edge from component node c to shared function node s. For each element $s \in$ sfused(t) there is an edge from shared function node s to test node t.

[0057] Given the above definitions, it is now possible to give the diagnosis algorithm according to the invention. The algorithm is simply to compute and sort *posteriori* likelihoods of component configurations given test results. Let $\pi \subseteq \psi$ be the passed tests. Let $\varphi \subseteq \psi$ be the failed tests. Bayes' Rule gives Equation (9).

[0058] All of these conditional probabilities will be normalized by the same quantity P (π, φ). This quantity is the prior probability of the test results and is difficult to compute. So the invention uses the likelihood of Equation (10).

[0059] The only nontrivial quantity to compute is P (π, φ | C, $\bar{C}$). If there are no shared functions, this is easy and leads to Equation (11), where P (π | C, $\bar{C}$), the probability of the passed tests given the test results, is given in Equations (12)-(14), and P (φ | C, $\bar{C}$), the probability of the failed tests given the test results, is given in Equations (15) and (16).

[0060] If there are shared functions, then use the law of total probability of Equation (17), where the first factor in the summand is in turn a product of factors computed according to Equations (7) and (8) as given in Equation (18).

[0061] The conditional probabilities of the shared function states are computed exactly like the test result probabilities of Equation (11) as given in Equations (19)-(21).

Improving Computation Time

[0062] Diagnosis could be performed by straightforward evaluation of Equations (10) through (17) for each possible state of the components and shared functions. However, that approach would take

$$O(2^{|\Phi|+|\Omega|})$$

time, which is clearly unacceptable for most practical applications. According to the invention, techniques for reducing the computation time can be applied, the most important of which are:

- reducing the number of candidate diagnoses, i.e., of component states $(C, \bar{C})$ for which *posteriori* likelihood Equation (10) is computed, and
- reducing the time required to evaluate Equation (17) by eliminating states of the shared function power set which do not affect the sum.

a) Reducing the number of candidate diagnoses

[0063] First, let's consider heuristics for reducing the number of component states. This can be achieved by making a reasonable assumption concerning the maximum number of simultaneously failed components. The invention assumes that component failures are independent. So unless the prior probabilities of failure are large, multiple failures are rare. This observation suggests choosing a maximum number of simultaneous failures N and computing Equation (10) only for those $C \subseteq \Phi$ with $1 \leq |C| \leq N$. This is the strategy preferably used by the invention.

[0064] Another strategy is that used by the aforementioned HP FaultDetective, is based on Occam's Razor: postulate only as many failed components as necessary. In other words, take N = 1 and compute the likelihoods. If any likelihood is nonzero, stop. Otherwise increase N by one and repeat. This way, a set of diagnoses is found with the minimum cardinality necessary to explain the test results. There are two dangers to this approach:

1. In pathological situations where unmodeled dependencies exist between tests, the algorithm may not stop in a reasonable amount of time. This can occur for example when a test fixture is set up incorrectly.

2. The Bayesian algorithm produces a nonzero likelihood for a diagnosis if has any chance whatsoever. A likelihood threshold would have to be set, but it is hard to set when the hard-to-determine denominator is being omitted from Equation (9).

[0065] This strategy works well with the HP FaultDetective but does not work well with the invention, because the invention can find candidate diagnoses with extremely small likelihoods. Even when $|C| = 1$, the invention will find some diagnoses with small likelihoods, for example 10^{-50} or even 10^{-100}.

b) Active shared functions

[0066] Now let's consider the problem of reducing the size of the power set $K(\Omega)$ over which Equation 17 is summed. It is evident that a shared function plays a role in diagnosing only those components over which it has coverage, and only when at least one conducted test makes use of the shared function. Therefore, Equation 17 may be summed over the much smaller power set of Equation (21.1), where Ω is the active shared function set as defined in Equation (22), which uses the provisional active shared function set, which is defined as in Equation (22.1).

[0067] The restriction to $K(\tilde{\Omega})$ is justified in the Equations because the states of $K(\Omega)$ can be paired relative to any shared function s, so that the members of each pair are identical except for s passing in one and failing in the other. If s is not used by any test in $(\pi \cup \varphi)$, then Equation (22.2) is invariant for the pair, and the sum of Equation (22.3) yields the probability of the other components of the state. So summing the pairs causes s to drop out of for the purposes of Equation (17).

[0068] As for the restriction of $\hat{\Omega}$, consider Equation (20). If a shared function $s \in \sigma$ has no coverage on any presumed faulty component $c \in C$, then sfcov(s, c) is uniformly zero, implying that the innermost product in Equation (20) is 1 for that s. This forces a factor of zero in the outermost product, making Equation (22.4). That result backs through

Equation (19) into Equation (17), making the whole term zero. Thus, no state term need be evaluated which posits failure for such a shared function. And again, if a state posits that the shared function succeed, it will simply cause a "1" to be factored into the product of Equation (21). So there is no reason to include that shared function in the state over which Equation (17) is summed.

[0069] The provisional active shared function set$\tilde{\Omega}$ can be quickly computed once at the beginning of the diagnosis, since it depends only on the tests which have been conducted. If the conducted tests are few relative to the available tests, this can effect a considerable reduction in the number of shared functions under consideration. The active shared function set Ω is winnowed from this separately for each combination of faulty components to be evaluated. Limiting the number of simultaneous faults to be considered (cf. above) usually produces a major reduction in the size of this set.

[0070] Some examples with the number of active shared functions for different models are shown in **Table 2.** The first four columns of the table give the name of the model, and the number of components, tests, and shared functions in the model. Column 5 shows the maximum number of active shared functions for which the state has been observed to be expanded. That many active shared functions are not always encountered. Column 7 gives the average size of the power set over which Equation 17 is expanded, which is the average of

$$2^{\#\text{active SFs}}.$$

[0071] This is the computational time factor paid for handling the shared functions. Column 6 is the base-2 log of column 7, giving the effective "average" number of active shared functions. The Boise data set is for a disk drive controller board, and Lynx3 is a board from a PC. The observed figures for them were derived over many runs of actual test results, and the effective average SF figures were almost always the same to the second decimal. Cofxhfdf is a model of a spectrum monitoring system, a small building full of radios, measurement equipment, and the cabling between them. The figures in the table were derived by arbitrarily having the first 30 tests fail and the next 30 tests pass. This is an artificial test result, but such large numbers of test failures do occur for the spectrum monitoring system. The result is encouraging, for the expansion factor of 5.85 is nowhere near 2^{203}. Running that diagnosis took 7.9 seconds of real time on a 200MHz Pentium Pro computer, which includes the time for loading and starting the program, and reading in the model. The program is written in Java.

c) Short Circuiting

[0072] The first product of Equation (18) can be zero if a passed test with 100% coverage clears an assumed bad component. It is actually bad form for a model to claim 100% coverage, so it may not be worthwhile to check for this. A more interesting case is that a term of the second product is zero. This means that no assumed-bad component could have caused one of the failed tests to fail. It is worth checking for this condition to avoid needless processing.

d) Factoring

[0073] In computing the sum of Equation (17), its first factor expands according to Equation (18), of which the first product is computed according to Equation (7). This in turn contains the factor of Equation (22.5), which is invariant over all the terms of the sum, and can therefore be pulled out of the loop.

e) Miscellaneous

[0074] The above speedups reduce the order of complexity of the algorithm. Other programming techniques also serve to reduce the required processing time. For example, coverages of failed tests must be matched against failed components. This goes faster if components, tests, and coverages are kept sorted. Bitmaps can be used to compute set intersections or unions, as for winnowing out the active shared function set. But the active shared function set should be kept in a squeezed representation for enumerating all of its states. It is well to pre-allocate arrays where possible, to avoid allocating them and freeing them during execution.

Conclusions

[0075] It will be apparent to those skilled in the art from the detailed description and the following procedures that a diagnosis engine constructed according to the present invention will require an amount of memory less than the amount of memory required to store the model and the amount of memory required to store the output multiplied by a small factor which is a constant independent of the model and output sizes. This makes such a diagnosis engine well suited to use as an embedded diagnosis engine.

[0076] The effect of obtaining the low memory consumption comes from using equations used to exactly compute the conditional likelihood of the component failures given the test results. Using the above equation numbering, this would be Equation 10, which contains values that must be computed from Equation 17 (which in turn uses Equation 18, which in turn uses Equations 19, 20, and 21) and the independence equation (0.1). However, it is clear that the content of those equations can be expressed also by other equations without departing from the scope of the present invention.

[0077] The invention allows that little memory is needed to compute the diagnosis in addition to that needed to store the model and the output. For better illustration, it shall be identified what additional memory is needed to compute the diagnosis. The effect of low memory consumption comes from the features that use that additional memory.

[0078] Computing the values of the left-hand sides of these equations 10 and 17-21 from the right hand sides does not require the storage of any intermediate results other than:

- a floating point register or memory location to accumulate the sum in Equation 17,
- a floating point register or memory location to accumulate the products in Equation 18,
- two floating point registers or memory locations to accumulate the products in Equation 20,
- one floating point register of memory location to accumulate the products in Equation 21.

[0079] That means that minimum 4 floating point registers or 4 memory locations are needed for calculating the equations 10 and 17-21.

[0080] In order to improve computation time, the Active Shared Functions (Equation 22) can be applied. This, however, increases the amount of memory needed in order to compute and store the Active Shared Functions. There are two objects that must be stored in memory: the Provisional Active Shared Functions and the Active Shared Functions.

[0081] The Provisional Active Shared Functions are normally computed once, typically before the computation is started. The Provisional Active Shared Functions are a subset of the Shared Functions. One way to store the Provisional Active Shared Functions is as an array of integers, where each integer in the array gives the index of a Shared Function that is a Provisional Active Shared Function. So, the Provisional Active Shared Functions can be stored in p integer sized memory locations, where p is the number of Provisional Active Shared Functions, which is less than the number of Shared Functions. The Active Shared Functions change during the course of the diagnosis computation. However, there is only one set of Active Shared Functions at any one time. The Active Shared Functions are a subset of the Provisional Active Shared Functions. So, the Active Shared Functions can be stored in no more integer memory locations than the number of Shared Functions.

[0082] In a nutshell, this means that the effect of small memory consumption comes from the direct and exact evaluation of statistical equations, such as Equations 10 and 17-21. Computing the values of the left-hand sides of these equations requires only a few floating point registers/memories. In order for this evaluation to be performed more efficiently, Provisional Active Shared Function and Active Shared Function sets can also be computed. These sets each require no more integer memory locations than the number of Shared Functions in the model. Thus, the number of temporary memory locations needed to compute the diagnosis grows linearly with the number of Shared Functions. To obtain the overall memory requirement, memory to store the model and the output of the diagnosis must also be added.

Illustrative example

[0083] The effect of the invention can also be explained in more descriptive way. Among searching methods that seek the best of a large number of combinations, there are two principal variants: those that search depth-first, and those that search breadth-first. As a more pictorial example, it shall be assumed that the largest apple on the tree is to be found.

[0084] For the ***depth-first search,*** one will go up the trunk to the first branch and follow that branch. When that branch divides, one will follow the larger subbranch, and so on, each time following the larger subbranch. One will eventually come to an apple, or to a leaf, or an end of a twig. If it is an apple, its position is jotted down and sized on a slate. Then one will go back down to the base of that last branch and explore up the alternative branch. If one will ever find an apple that is bigger than the one noted on the slate, the slate will be erased and the position and size of the new apple is noted. Eventually, the whole tree will have been explored, and the slate never had to record more than one apple's position and size. It has to be kept keep track of where one has been, but that doesn't take too much memory. All what is required is a list of the form: 1st layer of branches: 3rd alternative; 2nd layer: 2nd alternative; and so on. If the tree has no more than ten layers of branches upon branches upon branches, one will only have to keep a list of ten entries.

[0085] It is clear that this procedure requires a certain amount of time. Most likely, the biggest apple is on or near one of the bigger, low-hanging branches. To exploit this, one will do a ***breadth-first search.*** The size of the first layer of branches is surveyed. It will be started by looking for apples on the largest, and the branches will be followed looking for apples. But if it ever gets so far out in the bushiness that the branch one is on is smaller than some other in the survey, a note of the present location is made and that other branch will be explored. This way, one will always exploring the largest known previously-unexamined branch. A note is kept of the biggest apple so far. If one will ever come to a branch which is too small to support an apple of that size, that branch needs not be explored any farther, nor any of its subtwigs. This builds a fast-growing list of branches which one will might need to come back to, but the reward is that one will always look in the most likely places.

[0086] The invention thus minimizes the amount of storage required because it does a depth-first search. In order to improve computation time, the invention can apply a breadth-first search (corresponding an application of shared functions) in that it "looks at the tree" and finds that "most of the boughs are dead and barren of leaves and fruit", so it doesn't bother traversing them. And once it is up in the tree, it keeps avoiding dead branches, and ignoring grafted-on branches from orange trees.

Computer code examples

[0087] The three procedures in the attachments outline in words examples of computer code that could be used to implement the present invention. The first Procedure Diagnose of Attachment (a) gives the method for computing the likelihood of component failures using the speed improvements described in a) and b) of the section 'Improving Computation Time'. Procedure evalDiagnosis of Attachment (b) and Procedure findSfPEsc of Attachment (c) are used by the Procedure Diagnose.

****

Attachment (a): Procedure Diagnose

Parameters:

- model
- passed tests array
- failed tests array

Produces

- list of possible diagnoses, sorted in descending order of likelihood

```
1. Generate provisionalActiveSFs, an array of integers. This contains, in ascending order, the identification number of each shared function, which is depended on by any test in the passed and failed test arrays.
2. Create the diagnosis list as an empty set.
3. for N := 1 ... maximum number of simultaneous faults to be considered
   a. for C (the component set) running through all combinations of N faulty components:
      i. Generate activeSFs, an array of integers. This contains the subset of provisionalActiveSFs, which depend on any component in C.
      ii. Evaluate the likelihood of C (and only C) being the failed components, by calling evalDiagnosis, giving it C and activeSFs.
      iii. If the likelihood > 0, then make an entry in the diagnoses list, containing C and its associated likelihood.
4. Sort the diagnosis list in order of descending likelihood.
5. Return the diagnosis list.
```

****

Attachment (b): Procedure evalDiagnosis

Parameters

- model
- passed tests array
- failed tests array
- C, the list of assumed-bad components
- activeSFs, the array of shared functions which both depend on a component in C, and are used by a passed or failed test.

Produces

- a number giving the likelihood that the pattem of passed and failed tests could have been caused by the failure of all the components in C, and no others.

1. Call the number of bits in an integer memory location b. If activeSFs has more than b elements, signal an error. (In the preferred embodiment b=32 because the preferred embodiment uses the Java language which uses 32 bit integers.) .

2. Compute Pprior := the prior probability that all components in C fail while all others succeed. (This is the product of the individual prior probabilities.)

3. Compute sfPEscape := an array of numbers with an entry for each shared function, giving the probability that the shared function will pass given bad components C. (See procedure findSfPEsc.)

4. Set sumProb := 0.

5. for sfPattern := 0 to 2^(#active SFs) –1 (We interpret the integer sfPattern as a small array of bits: counting from the right, if bit i is 1,then the i-th active shared function fails. Otherwise, the shared function passes. This is actually shared function number activeSFs[i].)

    a. Compute PSFPat := the probability of occurrence of this pattern, by multiplying together the separate probabilities of the active shared functions. Those probabilities are:

        i. 1-sfPEscape[activeSFs[i]], if bit i is 1 in sfPattern

```
        ii. sfPEscape[ activeSFs[i] ], if bit i is 0 in sfPattern
    b. Compute condPofPassed := probability that all passed tests ought to have passed, given the bad components C and the failed shared functions indicated by sfPattern.
    c. Compute condPofFailed := probability that all failed tests ought to have failed, given the bad components C and the failed shared functions indicated by sfPattern.
    d. Add PSFPat * condPofPassed * condPofFailed to sumProb.
6. Return Pprior * sumProb.
```

****

Attachment (c): Procedure findSfPEsc

Parameters

- model
- C, an array of assumed-bad components
- activeSFs, an array of active shared functions.

Produces

- an array of numbers with as many entries as there are shared functions in the model (not just active shared functions), giving the probability that the shared function will pass, given bad components C.

```
1. sfPEscape := new numeric array with one entry for each shared function (not just active, but all of them).
2. Set each element of sfPEscape to 1.
3. For s := each element in activeSFs
    a. For c := each component covered by s
        i. If c is in C, then multiply (1-sfcoverage(s, c)) into sfPEscape[s]
4. Return sfPEscape.
```

Claims

1. A diagnosis engine for diagnosing a device with a plurality of components, wherein:

   states of the components are assumed to be probabilistically independent, for computing the probability of any particular set of components being bad and all others being good,

   states of shared functions, applicable for testing the functionality of some components in the same way, are assumed to be probabilistically independent given component states, for computing the probability of any particular set of shared functions being failed and another particular set of shared functions being passed given that a particular set of components are bad, and

   states of tests applicable on the device are assumed to be probabilistically independent given component and shared function states, for computing the probability of any particular set of tests being failed and another particular set of shared functions being passed given that a particular set of components are bad, and the rest are good, and a particular set of shared functions are failed, and the rest are passed;

   the diagnosis engine receiving:

   test results of a set of tests on the device where at least one test has failed, and

   a model giving the coverage of the tests on the components of the device and information describing probabilistic dependencies between the tests; and

   the diagnosis engine comprising:

   means for setting or specifying a number N of components which may be simultaneously bad, and

   computing means for computing the likelihood that each of subsets of the components with size less than or equal to N are the bad components, whereby the computation is substantially exact within floating point computation errors.

2. A diagnosis engine for diagnosing a device with a plurality of components, wherein:

   states of the components are assumed to be probabilistically independent, for computing the probability of any particular set of components being bad and all others being good,

   states of shared functions, applicable for testing the functionality of some components in the same way, are assumed to be probabilistically independent given component states, for computing the probability of any particular set of shared functions being failed and another particular set of shared functions being passed given that a particular set of components are bad, and

   states of tests applicable on the device are assumed to be probabilistically independent given component and shared function states, for computing the probability of any particular set of tests being failed and another particular set of shared functions being passed given that a particular set of components are bad, and the rest are good, and a particular set of shared functions are failed, and the rest are passed;

   the diagnosis engine comprising:

   means for specifying the component prior probabilities, coverages, and shared function coverages,

   means for specifying which tests have passed or failed or which were not performed,

   means for setting or specifying a number N of components which may be simultaneously bad, and

   computing means for computing the likelihood that each of subsets of the components with size less than or equal to N are the bad components, whereby the computation is substantially exact within floating point computation errors.

3. .The diagnosis engine of claim 1 or 2, further comprising:
means of outputting for one or more of the components the likelihood that the component is bad.

4. The diagnosis engine of claim 1 or 2, further comprising:
means for setting a default value for the number N of components which may be simultaneously bad, whereby the default value preferably is 1 or 2.

5. The diagnosis engine of claim 1 or 2, wherein the state of each component is either good or bad, and/or the state of each shared function is either passed or failed, and/or the state of each test is passed or failed.

6. The diagnosis engine of claim 1 or 2, wherein the amount of memory required by the engine during the course of a diagnosis is less than the amount of memory needed to store the model and the amount of memory needed to store the output through multiplication by a constant factor.

7. The diagnosis engine of claim 1 or 2, wherein the computing means computes the failure likelihood based on Equation (10) in combination with Equations (17) to (21).

8. The diagnosis engine of claim 7, comprising:

a floating point register or memory location to accumulate the sum in Equation (17),

a floating point register or memory location to accumulate the products in Equation (18),

two floating point registers or memory locations to accumulate the products in Equation (20), and

one floating point register of memory location to accumulate the products in Equation (21).

9. The diagnosis engine of claim 1 or 2, further comprising means for computing shared functions, preferably Provisional Active Shared Functions and the Active Shared Functions.

10. The diagnosis engine of claim 9, wherein the means for computing shared functions compute the shared functions based on Equation (22).

11. A method for diagnosing a device with a plurality of components, whereby:

states of the components are assumed to be probabilistically independent, for computing the probability of any particular set of components being bad and all others being good,

states of shared functions, applicable for testing the functionality of some components in the same way, are assumed to be probabilistically independent given component states, for computing the probability of any particular set of shared functions being failed and another particular set of shared functions being passed given that a particular set of components are bad, and

states of tests applicable on the device are assumed to be probabilistically independent given component and shared function states, for computing the probability of any particular set of tests being failed and another particular set of shared functions being passed given that a particular set of components are bad, and the rest are good, and a particular set of shared functions are failed, and the rest are passed;

the method comprising the steps of:

receiving test results of a set of tests on the device where at least one test has failed, and

receiving a model giving the coverage of the tests on the components of the device and information describing probabilistic dependencies between the tests,

setting or specifying a number N of components which may be simultaneously bad, and

computing the likelihood that each of subsets of the components with size less than or equal to N are the bad components, whereby the computation is substantially exact within floating point computation errors.

12. A software product, preferably stored on a data carrier, for executing the method of claim 10 when run on a data processing system such as a computer.

13. A software program, adapted to be stored on or otherwise provided by any kind of data carrier, for executing the steps of the method of claim 10 when run in or by any suitable data processing unit.

14. A diagnosis engine for diagnosing a device with a plurality of components, wherein:

states of the components are assumed to be probabilistically independent, for computing the probability of any particular set of components being bad and all others being good, and

states of tests applicable on the device are assumed to be probabilistically independent given component states, for computing the probability of any particular set of tests being failed given that a particular set of components are bad, and the rest are good,;

the diagnosis engine receiving:

test results of a set of tests on the device where at least one test has failed, and

a model giving the coverage of the tests on the components of the device and information describing probabilistic dependencies between the tests; and

the diagnosis engine comprising:

means for setting or specifying a number N of components which may be simultaneously bad, and

computing means for computing the likelihood that each of subsets of the components with size less than or equal to N are the bad components, whereby the computation is substantially exact within floating point computation errors.

Table 1: Summary of notation

Φ	Set of components
Ψ	Set of tests
Ω	Set of shared functions
c	A component
t	A test
s	A shared function
C	A set of components, $C \subseteq \Phi$
T	A set of tests, $T \subseteq \Psi$
S	A set of shared functions, $S \subseteq \Omega$
π	The passed tests, $\pi \subseteq \Psi$
ϕ	The failed tests, $\phi \subseteq \Psi$
$\mathrm{sfcov}(s, c)$	Coverage of shared function s on component c
$\mathrm{cov}(t, c)$	Coverage of test t on component c
$\mathrm{sfused}(t)$	Set of shared functions used by test t
$\mathrm{sfvar}(s)$	Variability of shared function s
$\mathrm{sfprob}(s)$	Probability that a test t with $s \in \mathrm{sfused}(t)$ fails due to s failing
$\kappa(A)$	The set of configurations of a set of random variables A
N	Assumed maximum number of simultaneously failed components
M	Number of elements in a set partition
$\ni$	"Such that"
$\|A\|$	Cardinality of the set A
$\overline{A}$	Complement of set A

Model	#components	#tests	#SFs	aSFs max	aSFs eff. avg.	SF expansion
Boise	44	40	6	4	3.48	11.15
Lynx3	53	67	3	3	1.94	3.83
Cofxhfdf	284	184	203	6	2.55	5.85

Table 2

$$P(S \subseteq \Omega \text{ fail} \mid C \subseteq \Phi \text{ bad}, \overline{C} \text{ good}) = \prod_{s \in S} P(s \text{ fail} \mid C \subseteq \Phi \text{ bad}, \overline{C} \text{ good}) \tag{1}$$

$$\begin{aligned} P(s \in \Omega \text{ fail} | C \subseteq \Phi \text{ bad}, \overline{C}\text{good}) &= 1 - \prod_{c \in C} (1 - \text{sfcov}(s, c)) && (2) \\ &= 1 - \prod_{\substack{c \in C \\ \text{sfcov}(s,c) \neq 0}} (1 - \text{sfcov}(s, c)) && (3) \end{aligned}$$

$$P(T \subseteq \Psi \text{ fail} \mid C \text{ bad}, \overline{C} \text{ good}, S \text{ fail}, \overline{S} \text{ pass}) = \prod_{t \in \Psi} P(t \text{ fail} \mid C, \overline{C}, S, \overline{S}) \tag{4}$$

$$\begin{aligned} P(t \in \Psi \text{ fail} \mid C \subseteq \Phi \text{ bad}, \overline{C} \text{ good}) &= 1 - \prod_{c \in C} (1 - \text{cov}(t, c)) && (5) \\ &= 1 - \prod_{\substack{c \in C \\ \text{cov}(t,c) \neq 0}} (1 - \text{cov}(t, c)) && (6) \end{aligned}$$

$$\begin{aligned} &P\left(t \in \Psi \text{ passed } \middle| C \subseteq \Phi \text{ bad}, \overline{C} \text{ good}, S \subseteq \Omega \text{ failed}, \overline{S} \text{ passed}\right) \\ &\quad = \prod_{c \in C} (1 - \text{cov}(t, c)) \prod_{s \in \text{sfused}(t)} (1 - \text{sfprob}(s) P(s \text{ failed} \mid C, \overline{C}, S, \overline{S}))) \end{aligned} \tag{7}$$

$$\begin{aligned} &P(t \in \Psi \text{ failed} \mid C \subseteq \Phi \text{ bad}, \overline{C} \text{ good}, S \subseteq \Omega \text{ failed}, \overline{S} \text{ passed}) \\ &\quad = 1 - \prod_{c \in C} (1 - \text{cov}(t, c)) \prod_{s \in \text{sfused}(t)} (1 - \text{sfprob}(s) P(s \text{ failed} \mid C, \overline{C}, S, \overline{S}))) \end{aligned} \tag{8}$$

$$P(C \text{ bad}, \overline{C} \text{ good} \mid \pi \text{ pass}, \phi \text{ fail}) = \frac{P(\pi \text{ pass}, \phi \text{ fail} \mid C, \overline{C}) P(C, \overline{C})}{P(\pi \text{ pass}, \phi \text{ fail})} \tag{9}$$

$$L(C \text{ bad}, \overline{C} \text{ good} \mid \pi, \phi) = P(\pi \text{ pass}, \phi \text{ fail} \mid C, \overline{C}) P(C, \overline{C}). \tag{10}$$

$$P(\pi, \phi \mid C, \overline{C}) = P(\pi \text{ pass} | C, \overline{C}) P(\phi \text{ fail} | C, \overline{C}) \tag{11}$$

$$\begin{aligned} P(\pi \text{ pass}|C,\overline{C}) &= \prod_{t\in\pi} P(t \text{ pass}|C,\overline{C}) && (12)\\ &= \prod_{t\in\pi}\prod_{c\in C} P(t \text{ pass}|c) && (13)\\ &= \prod_{t\in\pi}\prod_{c\in C} (1-\text{cov}(t,c)) && (14) \end{aligned}$$

$$\begin{aligned} P(\phi \text{ fail}|C,\overline{C}) &= \prod_{t\in\phi} P(t \text{ failed}|C,\overline{C}) && (15)\\ &= \prod_{t\in\phi}\left(1-\prod_{c\in C}(1-\text{cov}(t,c))\right) && (16) \end{aligned}$$

$$P(\pi,\phi \mid C,\overline{C}) = \sum_{(\sigma \text{ failed},\, \overline{\sigma} \text{ passed})\in\kappa(\Omega)} P(\pi,\phi|C,\overline{C},\sigma,\overline{\sigma})P(\sigma \text{ failed}, \overline{\sigma} \text{ passed}|C,\overline{C}) \qquad (17)$$

$$P\left(\pi,\phi \middle| C,\overline{C},\sigma,\overline{\sigma}\right) = \prod_{t\in\pi} P(t \text{ passed} \mid C,\overline{C},\sigma,\overline{\sigma}) \prod_{t\in\phi} P(t \text{ failed} \mid C,\overline{C},\sigma,\overline{\sigma}). \qquad (18)$$

$$P(\sigma \text{ fail}, \overline{\sigma} \text{ pass}|C,\overline{C}) = P(\sigma \text{ fail}|C,\overline{C})P(\overline{\sigma} \text{ pass}|C,\overline{C}) \qquad (19)$$

$$P(\sigma \text{ fail}|C,\overline{C}) = \prod_{s\in\sigma}\left(1-\prod_{c\in C}(1-\text{sfcov}(s,c))\right) \qquad (20)$$

$$P(\overline{\sigma} \text{ pass}|C,\overline{C}) = \prod_{s\in\overline{\sigma}}\prod_{c\in C}(1-\text{sfcov}(s,c)) \qquad (21)$$

$$\hat{\Omega} = \left\{ s \in \tilde{\Omega} \mid \exists c \in C \ni \text{ sfcov } (s,c) > 0 \right\} \qquad (22)$$

$$P(C\,\text{bad},\ C \subseteq \Phi) = \prod_{c \in C} P(c\,\text{bad}) \tag{0.1}$$

$$P(s \in \Omega\ \text{failed}|c \in \Phi\ \text{bad}) = \text{sfcov}(s, c) \tag{0.2}$$

$$P(t \in \Psi\ \text{failed}|c \in \Phi\ \text{bad}) = \text{cov}(t, c) \tag{3.1}$$

$$\text{sfprob}(t) = 1 - \text{sfvar}(t)/2 \tag{6.1}$$

$$(\sigma\,\text{failed}\ ,\overline{\sigma}\,\text{passed}) \in \kappa\left(\hat{\Omega}\right) \tag{21.1}$$

$$\tilde{\Omega} = \bigcup_{t \in \pi, \phi} \text{sfused}\ (t) \tag{22.1}$$

$$P(\pi, \phi\,|\,C, \overline{C}, \sigma, \overline{\sigma}) \tag{22.2}$$

$$P(\sigma\ \text{failed},\ \overline{\sigma}\,\text{passed}\,|\,C, \overline{C}) \tag{22.3}$$

$$P\left(\sigma\ \text{fail}\,|C, \overline{C}\right) = 0 \tag{22.4}$$

$$\prod_{c \in C} (1 - \text{cov}\ (t, c)) \tag{22.5}$$